



专题:基于 Android 系统的终端安全

Android 安全的研究现状与展望

卿斯汉^{1, 2, 3}

(1. 中国科学院软件研究所, 北京 100190;

2. 北京大学软件与微电子学院, 北京 102600;

3. 中国科学院信息工程研究所信息安全国家重点实验室, 北京 100093)

摘要: Android 是目前最受用户欢迎的智能手机操作系统, 与此同时, 其安全态势也日益严重。介绍了 Android 版本更新的发展历程、Android 系统的特色、Android 的安全机制、Android 的安全隐患分析、Android 恶意软件与攻击、Android 的分析与防御。最后, 阐述了 Android 安全的研究现状与发展趋势以及今后可能的研究方向。

关键词: Android 系统; 安全机制; 安全隐患; 恶意软件; 分析与防御

中图分类号: TP301

文献标识码: A

doi: 10.11959/j.issn.1000-0801.2016256

Research status and outlook of Android security

QING Sihan^{1, 2, 3}

1. Institute of Software, Chinese Academy of Sciences, Beijing 100190, China

2. School of Software and Microelectronics, Peking University, Beijing 102600, China

3. State Key Laboratory of Information Security, Institute of Information Engineering,
Chinese Academy of Sciences, Beijing 100093, China

Abstract: Currently, Android is the most popular operating system for smartphones. At the same time, its security situation is becoming increasingly serious. The following topics: the development course of Android version updates, the features of Android system, Android security mechanism, analysis of Android security risks, Android malware and attack, and the analysis and defense measures were covered. Finally, the current research status and progress, and the future research directions regarding Android security were addressed.

Key words: Android system, security mechanism, security risk, malware, analysis and defense

1 引言

Android 是目前最受用户欢迎的智能手机操作系统。根据 Gartner^[1]的统计数据, Android 在 2015 年第 4 季度的市场占有率已经高达 80.7%, iOS 仅占有 17.7% 的市场份

额, 其余如 Windows、黑莓等则远远落后, 具体见表 1。

与此同时, Android 的安全态势也日益严重。为了适应不断变化的市场需求, 并不断改善手机的安全性能, Android 系统的版本更新速度很快。表 2 是 Android 版本的更新历史, 包括不同的版本代码、版本号、首次发行日期以

收稿日期: 2016-07-12; 修回日期: 2016-09-28

基金项目: 国家自然科学基金资助项目 (No.61170282)

Foundation Item: The National Natural Science Foundation of China (No.61170282)

表 1 移动操作系统市场占有率

操作系统	2015 年第 4 季度/千台	2015 年第 4 季度市场占有率	2014 年第 4 季度/千台	2014 年第 4 季度市场占有率
Android	325 394.4	80.7%	279 057.5	76.0%
iOS	71 525.9	17.7%	74 831.7	20.4%
Windows	4 395.0	1.1%	10 424.5	2.8%
黑莓	906.9	0.2%	1 733.9	0.5%
其他	887.3	0.2%	1 286.9	0.4%
合计	403 109.4	100.0%	367 334.4	100.0%

表 2 Android 版本更新历史

版本代码	版本号	首次发行日期	API 级别
无	1.0	2008 年 9 月 23 日	1
	1.1	2009 年 2 月 9 日	2
Cupcake(纸杯蛋糕)	1.5	2009 年 4 月 27 日	3
Donut(甜甜圈)	1.6	2009 年 9 月 15 日	4
Eclair(松饼)	2.0~2.1	2009 年 10 月 26 日	5~7
Froyo(冻酸奶)	2.2~2.2.3	2010 年 5 月 20 日	8
Gingerbread(姜饼)	2.3~2.3.7	2010 年 12 月 6 日	9、10
Honeycomb(蜂巢)	3.0~3.2.6	2011 年 2 月 22 日	11~13
Ice Cream Sandwich(冰淇淋三明治)	4.0~4.0.4	2011 年 10 月 18 日	14、15
Jelly Bean(果冻豆)	4.1~4.3.1	2012 年 7 月 9 日	16~18
KitKat(奇巧巧克力)	4.4~4.4.4, 4.4W~4.4W.2	2013 年 10 月 31 日	19、20
Lollipop(棒棒糖)	5.0~5.1.1	2014 年 11 月 12 日	21、22
Marshmallow(棉花糖)	6.0~6.0.1	2015 年 10 月 5 日	23
Nougat(牛轧糖)	7.0	2016 年 8~9 月	24(可能)

及相应的 API 级别。

例如,当前最新的 Marshmallow 版本(API 23)主要聚焦对前一个版本(Lollipop 版本(API 21、API 22))的用户体验改进。主要的改进包括:引入一个新的权限架构;引入一个新的支持上下文的 API,可与 Google(谷歌)搜索 App 的新特性“Now on Tap(随时应用)”配合使用;引入一个新的电源管理系统,可以在不访问手机时减少设备的后台活动;增加对指纹识别的本地支持等。Lollipop 版本最重要的改进之一是:Google 推出了一种新的称为“质感设计(material design)”的设计语言,重新设计了用户界面。其特点是:可以在用户界面生成动画,动画是仿照真实的物理世界生成的。并且,该设计语言具有跨平台的特点。其他改进包括:对通知的改进(即使在锁屏时也能访问通知);Android 运行时(Android runtime, ART)正式取代 Dalvik 虚拟机,改善应用性能;优化电池使用等。又如,在 Android 2.3 版本后增加了基于硬件的 NX(no execute)支持,不允许在堆和栈中执行代码。在 Android 4.0 后,增加了“地址空间布局随机化(address

space layout randomization, ASLR) 功能,防止内存相关攻击。为防止内核级提权攻击,Android 4.3 后增加了具有强制访问控制机制的 SELinux 支持。

有时,Android 也会删除一些过时或有害的特征。例如,为防止“Setuid()-RLIMIT_NPROC 交互”型漏洞利用,Android 4.3 版本之后撤消了对 setuid()/setgid()的支持。又如,Android 4.3 版本未加声张地引入了一个名为 App Ops 的新特征,使用户可以有选择地禁止 App 安装时请求的某些权限,从而改进 Android 粗粒度的权限模型。但最终,Android 4.4.2 版本之后删除了这一新特性,因为 Google 认为这一试验性的特性会导致兼容性问题。

自 2008 年 9 月 23 日 Google 正式发布 Android 版本 1.0 以来,学术界和产业界对 Android 及其安全性进行了大量研究。除安全综述论文^[2-4]外,还有针对各类不同具体安全问题的讨论^[5-7]等。

本文将在参考文献[4]的基础上,结合近年来的最新研究成果^[8-12],对 Android 面临的安全问题与挑战进行全面的概括。



2 Android 系统特性

Android 是基于 Linux 内核的开源智能移动操作系统,由 Google 与开放手机联盟 (Open Handset Alliance, OHA) 合作开发。其特点是:开放性,相对于苹果系统的封闭性,Android 更加开放,具有更低的进入门槛,吸引了更多用户、第三方开发商与应用程序开发者;资源有限性设计,由于手机应用的资源(内存、电池等)有限,或者由于某些系统受到版权保护,必须进行相应的重新设计;简单性,为了最大程度地方便用户,Android 系统遵从“简单、快速和易用”的设计原则,在安全和用户体验之间进行了折中。

2.1 Android 系统的分层结构

Android 系统采用分层结构,由上到下分别是应用层、Android 中间件层和 Linux 内核层。每一层只专注于自己所需提供的服务,各层自己所需要使用的服务都由下面的层提供。

应用层既包含系统提供的基本应用,如桌面、拨打电话和通讯录等;也包含用户安装的第三方应用程序。

中间件层由应用框架、本地类和 ART 组成。其中,Android 应用框架是用 C/C++ 或 Java 编写的系统应用,由系统内容提供者(system content provider)和系统服务(system service)构成。开发者编写的 App 和 Android 系统提供的 App 调用应用框架,用于提高代码的复用性,并能大大简化代码的复杂度。本地类库是一组 C/C++ 库,提供诸如图形处理等基本功能。ART 由核心 Java 类库与 Dalvik 虚拟机(DVM)组成。

Linux 内核层提供最基本的服务,如进程调度、内存管理、网络协议栈、设备驱动等。同时,Linux 内核还作为硬件和软件栈之间的抽象层,将硬件透明化,提供有效的接口和函数,供上层调用,起到了承上启下的作用。Android 支持两种指令集架构:ARM 和 x86。

2.2 Android 的组件与组件间通信

Android 有 4 类组件:activity(活动)是 App 与用户之间进行交互的可视化界面;service(服务)是 Android 系统在后台运行的服务;content provider(内容提要)是一种 SQL 数据库,用于为 App 提供数据与数据共享支持;broadcast receiver(广播接收器)是用于接收广播消息的邮箱。

Android 的组件间通信主要通过 Intent 完成。Intent 是一个包含目标组件地址和数据的消息对象。Android API 定义了接收 Intent 和通过其中的信息启动 activity、service

和广播消息的方法。调用这些方法相当于通知 Android 系统开始调用目标应用中的执行代码。Intent 对象有两类:显式(标注目标组件名称)和隐式(不标注目标组件名称)。

2.3 Android 的进程通信机制

Android 在内核层实现了基于 Binder(OpenBinder 的精简版本)的轻量级进程间通信,称为组件间通信(inter-component communication, ICC),实现安全和高效率的进程通信。Binder 基于 C/S(客户端/服务器)通信模式,只需一次复制数据对象;可以自动传输发送进程的 UID/PID 信息,并支持实名和匿名 Binder。Binder 提供了远程过程调用(RPC)功能,概念上类似于 COM 和 CORBA 分布式组件架构,使一个进程可以类似本地调用地访问另一个进程所提供的功能。此外,Android 为 ICC 设计了一种特定的接口定义语言(Android interface definition language, AIDL),定义用于远程调用的方法和字段。例如:可以通过 ICC 调用绑定远程服务。

2.4 程序开发语言和 Java 本地接口

下述 4 种语言可以用于 Android 系统开发:汇编语言、C、C++ 和 Java。内核除含有少量汇编程序外,主要是 C 语言程序。某些原生程序和库是用 C++ 编写的。所有的其他应用,特别是定制程序,都是通过 Java 编写的。

通常,需要区分虚拟机编译的程序和为在特定计算平台(如 Intel x86 或 ARM)上运行而编译的程序,为一个特定平台编译的程序称为“本地(native)”程序。因为 Java 程序是通过它自己的字节码在虚拟机中执行的,所以不能直接执行本地程序。由于需要访问底层的操作系统机制(如内核调用),因此引入了 Java 本地接口(Java native interface, JNI),使 Java 程序可以执行库中用其他语言(如 C++)编译的代码。这么做具有重要的实际意义,但却降低了 Java 的安全性。

2.5 Dalvik 虚拟机

Dalvik 虚拟机(DVM)运行用 Java 编写的应用程序。Sun 公司开发的 Java 虚拟机(JVM)是基于栈的,因此可以在所有的硬件中运行。硬件与平台的无关性是 Java 设计的根本原则。由于移动设备的内在需求(如高性能与低能耗),DVM 采用了不同的设计原则:既要最大限度地与硬件平台无关,又要最大限度地提高性能和降低能耗。为了提高性能,DVM 是基于寄存器的,并非常适合 ARM 硬件。为了提高安全性,Java 检查变量的边界、封装

系统调用,并强制使用 JVM 定义的系统接口。但通过 JNI 的应用降低了 Java 的安全性,因为使用本地库可以执行旁路虚拟机的类型检查和边界检查,增加了栈溢出攻击的可能性。尽管如此,JNI 是进程间通信机制中的关键结构,因为 Binder 中间件是 C++ 库,必须通过 JNI 才能访问。

Dalvik 虚拟机有一套自身专用的指令集,并规定了这套指令集的指令格式与调用规范,称为 Smali 语言。

3 Android 的安全机制

Android 的安全机制一部分继承了 Linux 的安全机制,包括 POSIX 用户和 Linux 的权限机制;另一部分是 Android 特有的安全机制,主要包括:Android 沙盒、应用程序签名机制、组件封装机制、Android 本地库及运行环境安全机制、权限机制等。

每个应用程序都作为一个 Dalvik 虚拟机实例在自己的进程中运行。因此,Dalvik 虚拟机与 POSIX 用户安全机制一起构成了 Android 沙盒机制。

在 Android 系统中,每个 App 都被授予一个证书和相应的开发者私钥,用于区别不同的 App 开发者。因为 Android 使用 Linux 内核,故该证书可使 Android 系统判断 App 是否有权使用一些签名级的权限,并判断是否允许用户访问 Linux 标识为同一开发者的 App 文件。

Android 的组件封装机制可以保证 App 的安全运行。如果组件中的功能“exported”设置为“false”,则组件只能被 App 本身或拥有同一个 UID 的 App 访问,此时该组件被称为“私有组件”。如果“exported”设置为“true”,则组件可被其他程序调用或访问,此时该组件被称为“公开组件”。

Android 本地库及运行环境安全机制由内存管理单元(MMU)、强制类型安全和移动设备安全 3 个部分组成。MMU 为进程分配不同的地址空间,实施进程的内存空间隔离。Android 使用强类型 Java 语言,通过编译时的类型检查、自动存储管理和数组边界检查保证类型安全。同时,Binder 也是类型安全的。移动设备安全包括:认证(authentication)、授权(authorization)和记账(accounting),简称为“AAA”。

权限机制是 Android 最重要的安全措施之一,其特点可以概括如下。

- 遵循“最小特权”原则:如果 App 没有声明任何权限,就没有任何特权。

- 粗粒度,即“全有或全无(all-or-nothing)”的授权机制:App 对权限的申请和声明都被强制标识于 AndroidManifest.xml 清单文件中,通过<permission>、<permission-group>、<permission-tree>等标签指定。App 申请的权限在安装时提示给用户,用户做出同意或拒绝安装的决定。
- Android 系统中的权限可分为三大类:Android 手机所有者权限、Android root 权限和 App 权限。App 共有 4 类不同的权限保护级:正常级,只要申请即可使用,无需用户授权;危险级,安装时需用户同意才能使用;签名级;签名或系统级。其中,后两个权限保护级主要供系统使用。

4 Android 的安全隐患分析

4.1 Android 安全隐患的综合分析

在长期、高频率和大规模的使用中,面对复杂多变的应用环境,Android 的原生安全机制呈现出一系列漏洞和脆弱性。

Android 的安全模型及其生态系统包括但不限于下述安全隐患。

(1) Android 的开放性导致的安全隐患

- 存在大量 Android App 市场,且对 App 没有审核或只进行有限的审核。
- 对 Android App 的发行者缺乏背景审查。
- 可以不通过 Android App 市场安装 App,如通过 ADB 工具安装 App,成为另一个恶意软件的攻击面。

(2) Android 的粗粒度权限模型导致的安全隐患

- 安装时授权基于“全有或全无”机制:要么用户赋予 App 请求的全部权限,要么不允许 App 进行安装。因此,Android 用户可能由于不熟悉所请求的权限而同意安装 App。
- App 开发者经常请求超过实际需求的多余权限,突破了“最小特权”原则,产生所谓的“溢权(overprivilege)”问题。
- 不仅 Android 的权限管理机制是粗粒度的,许多 Android 权限也是粗粒度的。如,“Internet”是粗粒度权限,拥有该权限的 App 可以任意访问互联网资源。Android 系统不能实现基于域的 Internet 访问以及可选择地访问敏感资源,造成攻击者能够以合法身份滥用资源。



- 供开发者使用的官方权限文档不充分、不准确,有时还包含错误,容易造成开发者的失误,也可能导致安全威胁,如申请过多不必要的权限。
- 缺少运行时的权限撤消机制,导致运行时的“溢权”问题。

(3)与 Windows 和 Linux 等桌面环境相比较,Android App 更易受逆向工程攻击

其原因包括:与硬件无关的 Dalvik 字节码文件包含原始 Java 文件中的大量信息。此外,App 的用户界面布局与字符串通常以 XML 格式存储。因此,Android App 易于重包装,包括插入恶意软件。

(4)第三方库可以滥用其宿主 App 拥有的权限,此外,恶意宿主 App 可以篡改库,进行有害操作。

(5)Android 缺乏可配置的运行时 ICC 控制

- 无法防止 App 访问其他 App 的公开接口。与下述安全弱点(6)相结合,由于缺乏运行时 App 之间的访问控制,将会导致数据泄露与“困惑代理(confused deputy)”问题。该问题源于操作系统编译程序的“一仆二主”,即编译程序运行时同时具有来自两个调用的不同权限。
- 无法防止 App 截取 Intent 广播且终止随后的广播。通过截取系统事件广播,恶意 App 可以不被察觉地截取包含敏感信息的重要系统事件。
- 无法隔离 App 并防止 App 通过 ICC 和其他共享信道进行通信。这类弱点可以被共谋攻击所利用。

(6)Android ICC 的安全弱点可能导致泄露 App 的私有接口与敏感数据。

(7)App 可以通过 JNI 调用本地代码,从而导致安全问题。类似于安全弱点(4),第三方本地库可以滥用其宿主 App 拥有的权限。此外,恶意软件可以利用本地代码规避检测与监控工具的分析。

(8)Android 设备有多个标识符可以用作唯一的设备 ID,如 IMEI、硬件序列号、Android 系统 ID。因为 Android 设备容易泄露敏感信息,如果该设备 ID 被泄露,则攻击者可以追踪用户。

(9)Linux 层的原生脆弱性,如 Linux 内核及其本地库的漏洞构成 Android 系统与应用的安全隐患。

(10)设备制造商定制和预安装 App 中的漏洞构成安全隐患。

(11)App 的生命周期分为:安装、交互(包括与系统和

其他 App 之间的交互)和卸载 3 个阶段。当 App 被卸载后,属于该 App 的各种数据仍然在系统中广泛分布。由于 Android 未能彻底删除这些残余数据,将会产生所谓“数据残余”安全隐患。

(12)Android 中某些重要构件的缺陷会导致各种不同的安全隐患。如分组管理服务(package manager service, PMS)是一种用于安装、更新、配置和删除应用包的 Android 服务。它的设计缺陷可能会导致版本更新时的安全隐患。详情参看第 4.3 节中的阐述。

上述安全隐患对 Android 系统构成了严重威胁。除直接攻击硬件外,按照 Android 的分层结构,攻击者可以利用这些隐患对 Android 进行不同层次的攻击,包括 Linux 内核层、Android 系统核心程序层(Android 框架和 Android 本地库及运行时)和应用程序层。

在实践中,安全威胁并不拘泥于单一的攻击形式,往往是多种攻击形式的相互配合,如两个及以上程序的“合谋攻击”更加难以防范。

4.2 第三方库导致的 Android 安全隐患

在实践中,许多第三方库可以违反 App 开发者的意愿访问敏感资源,有下述 3 种不同的情形。

(1)滥用非官方公布权限的第三方库

App 开发者根据官方公布的文档(如开发者指南)知道一个库将使用哪些权限。但是,由于缺乏 App 内的权限隔离,当宿主 App 拥有某个(非官方公布)权限时,第三方库可以在没有相应权限的条件下“搭便车”访问敏感资源。如果第三方库动态检查它是否拥有特定权限或捕获某个安全异常,可以滥用权限而不事先通知用户。

(2)受污染的第三方库

虽然原生第三方库是合法的,但攻击者可以改写其二进制代码或源代码并重新发布。当 App 错误地使用这种受污染的库时,安装该 App 的用户将面临安全威胁。2013 年,一种称为 Uten.A 的恶意软件伪装成 Umeng SDK 库(一个移动分析平台),重新打包后发布。某些 Google Play 中提供的合法游戏 App 使用了该恶意软件,且许多用户安装了这些 App。受感染的移动设备会不受察觉地使用高价短信服务。

(3)易受攻击的第三方库

第三方库可以在运行时执行从 Internet 下载的类型或 JavaScript 代码。如果移动设备与不安全的网络相连接,且第三方库没有加密代码,则攻击者可以用恶意代码取代原有代码,并利用宿主 App 的权限泄露敏感信息。例如,如果

第三方库通过 addJavascriptInterface 向 JavaScript 暴露 getDeivceId 方法, 则 JavaScript 中的恶意代码可以读设备的 IMEI 码。

表 3 表明 5 个流行的第三方库的特征, 说明它们可以不动声色地请求敏感的 API (如短信、位置、Internet 等) 以及它们依赖 Java 语言动态特征的情形。在表 3 中, 第 3~6 列说明 App 潜在使用的权限 (“○”表示所需权限; “△”表示可选权限; “x”表示非官方公布的权限)。第 7~11 列说明第三方库运行时对 Java 语言动态特征的依赖性。

通过各种 Java 特性 (反射、动态类加载、类继承、本地方法、多线程、回调等) 实现的动态代码执行在运行时模糊了模块之间的边界。反射使程序开发者可以在运行时跨 Java 类检查或修改代码, 但同时也带来一系列安全隐患。例如, 恶意第三方库可以通过反射为原生的宿主模块生成代码, 并动态加载和运行生成的代码, 并以宿主模块的名义实施恶意操作。恶意第三方库甚至可以在运行时改变类名, 冒充成可信的宿主模块类。

表 3 的结果具有一般性。在真实世界, 20 个流行的第三方库中有 17 个使用非官方公布的权限; 295 个第三方库中有 72% 依赖于动态代码执行, 有 17% 使用 JNI。

4.3 系统更新导致的 Android 安全隐患

用户经常进行 Android 版本的更新操作, 会产生一系列安全隐患。

用户按下系统更新按钮之后, 更新镜像将从设备生产商的服务器中下载。该镜像包括 boot-loader.img、其他系统文件和一组在不同目录中分布的 Android 应用包文件。某些文件是旧系统中已有文件的补丁, 通常以 “.p” 为后缀, 如 radio.img.p。一旦收到该镜像, Android 设备重新启动。这时进入 “修复模式 (recovery mode)”, 认证该镜像并验证镜像的完整性。然后, 用新文件取代系统目录中已有的系统文件, 如启动程序、分组管理服务程序和 APK 等。最后, 设备以新操作系统重新启动, 并从不同的 Android 服务 (包括 activity 管理程序、服务管理程序、用户管理服务、分组管理

服务、输入管理程序等) 中继续更新系统组件。其中, 最值得关注的是分组管理服务 (PMS)。

PMS 是用于安装、更新、配置和删除应用包的 Android 服务。当用于更新时, PMS 在新操作系统的 /system/ 目录下安装或重新安装所有的系统 App, 然后在 /data/App 目录下安装或重新安装已有的第三方 App。当安装 App 时, PMS 注册其权限、共享 UID、activity、Intent 过滤器、服务等, 并需处理重复的属性 (如分组名和共享 UID) 和性质 (如权限)。这时, PMS 需要确定安装哪一个。为此, PMS 有一个名为 mSettings 的数据结构, 用于记录在旧操作系统中已有的 App 的所有相关信息, 包括一组类型为 HashMap 的列表, 如 mPackages、mUserIds、mSharedUsers、mPermissions 等。当需要安装新的系统分组或注册其性质时, PMS 首先查询这些列表, 确定在旧系统中是否存在其他具有相同属性或性质的分组。一旦发现新的分组或其性质在旧系统中已经存在, 则将基于分组或性质确定选择其中的一个。如当更新镜像中的 App 版本号高于已有的 App 版本号时, 则前者将取代后者。

为对用户屏蔽底层的实现细节, Android 具有层次化的体系结构。App 的属性和性质, 即权限、分组名、共享 UID 等是系统级的, 在系统中唯一地标识它的分组和权限。在存在重复的分组属性或性质的情况下, 更新过程就十分复杂。亦即, PMS 不仅需要检查有冲突的分组中的不同条件, 从而选择安装正确的一个, 还需要仔细地结合两个 App (如保留旧版本中的数据, 但使用新版本中的代码)。此时, 程序逻辑非常复杂, 因此容易出错, 一旦 PMS 选择了错误的属性或性质, 则会产生安全隐患。Android 经常通过名称引用特权、资源和数据。因此, 当安装或重新安装所有的 App 或重置所有的系统配置时, 与其系统副本具有相同名称的第三方的包属性或性质, 可能会获得其相应的系统属性或性质。此外当两个来自旧系统和新系统的 App 合并时, 如果旧系统中的 App 是恶意的, 则会产生安全隐患。

表 3 第三方库的特征

名称	分类	电话信息	位置	Internet	读/写短信	类加载	反射	回调	类继承	JNI
Facebook	社交	×	×	○		✓	✓	✓	✓	✓
Flurry	分析	×	○	○		✓	✓		✓	
Paypal	记账	×	×	○		✓			✓	
InMobi	广告		△	○	×	✓	✓	✓	✓	
Chartboost	广告	×		○		✓	✓		✓	✓



5 Android 恶意软件与攻击

Android 恶意软件呈快速增长的态势,我国的形势更加不容乐观。2016 年 2 月,著名的以色列信息安全公司 Check Point 发表最新报告:一种名为 HummingBad 的恶意软件,使中国受感染的 Android 手机超过 1 000 万台。HummingBad 采用一种名为“诱惑下载(drive-by download)”的攻击方法,当用户访问网站时手机就会受到感染。该恶意软件有两个组件:组件 1 通过漏洞利用试图获取 root 权限,如果失败则组件 2 发送伪造的系统更新通知,欺骗用户进行系统更新,尝试获得系统级特权。通过强制下载和点击广告,该恶意软件每月可以获得超过 30 万美元的收入。HummingBad 的背后是一家名为 Yingmob(微赢互动)的广告公司,开启了通过有组织的公司形式开发恶意软件的先例。

HummingBad 恶意软件的影响不仅局限于中国。除了最大的受害国中国和印度(受感染手机分别为 1 600 万台和 1 350 万台)外,其他依次为菲律宾、印度尼西亚和土耳其。美国受感染的手机为 288 800 台,英国和澳大利亚受感染的手机小于 10 万台。

下面以康夫机器人(Yasuo bot)恶意软件的攻击流程为例(如图 1 所示),说明恶意软件的典型工作方式。

5.1 Android 恶意软件的特征与分类

Drebin^[1]数据集中搜集了大量(至少 5 560 个恶意软件样本)真实的移动恶意软件。表 4 说明,恶意软件的公共特征通常包括:数据泄露、反射调用和动态类加载、滥用短信服务和本地代码利用。

从不同角度,可以对 Android 恶意软件进行分类。

(1)按照传统的方法

如木马、病毒、蠕虫、DDoS、劫持、僵尸、后门、间谍软件、恐吓软件、勒索软件、广告软件、跟踪软件等。

(2)通过恶意软件的目标与实现功能的不同

如僵尸网络功能;获取 root 权限或使用用户 root 其手

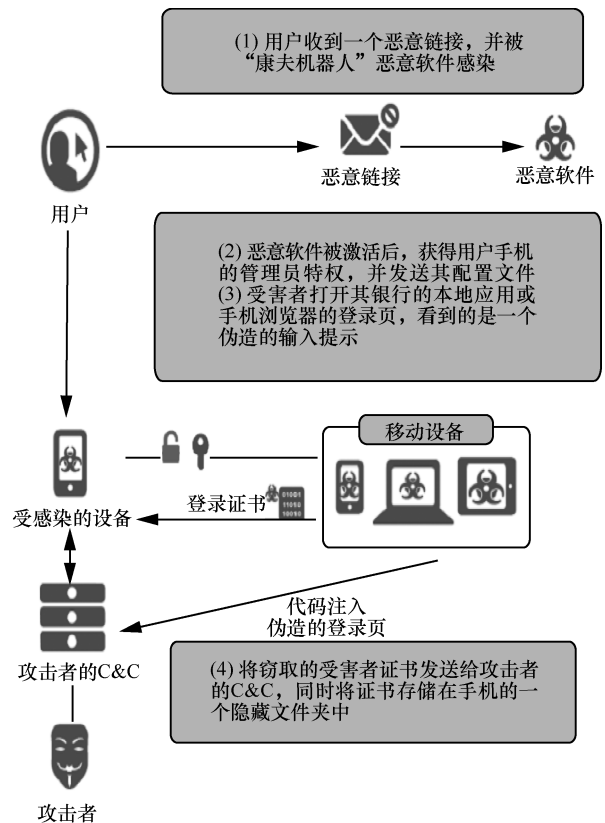


图 1 康夫机器人恶意软件的攻击流程

机;发送付费或恶意短信消息;盗窃用户银行信息;盗窃位置信息;安装其他 App 或二进制代码;安装黑客工具;感染相连的 Windows PC 的木马等。

(3)根据 Android 恶意软件的特征

主要包括:安装特征,如重打包、更新攻击、诱惑下载等;运行(activation)特征;恶意载荷(malicious payload)特征,如提权攻击、远程控制、付费欺骗、信息收集等;权限使用。

5.2 Android 的典型攻击方法

Android 常用的攻击方法包括但不限于以下几种。

(1)提权攻击

拥有较低(较少)权限的 App 访问拥有较高(较多)权

表 4 恶意 App 的特征

特征	恶意 App
数据泄露	AckPosts, Aks, Anesa, jSmsHider, Saiva, Vidro, Gonca, RootSmart, RATC, JSExploit-DynSrc, Xsider, Ssmasp, Mobsquz, FakeTimer, DroidKungFu, Spy.GoneSixty, Kmin, GGTrack, MobileTx, Dougalek, FakeDoc, Loozfon, Placms
滥用短信服务	MobileTx, Iconosys, UpdtKiller, Pirater, Mania, FakeInstaller, FakePlayer, Foncey
反射和动态类加载	Mobsquz, FakeDoc, FaceNiff, BaseBridge, DroidDream
本地代码	Anesa, Qicsom, RATC, DroidKungFu, Xsider, DroidSheep, Gmuse, FakeDoc, FaceNiff

限的组件而不受任何限制,称为提权攻击,是攻击 Android 最常用的一种方法。通常,将提权攻击分为“困惑代理”攻击和“合谋”攻击两大类。

在“困惑代理”攻击情形中,具有不同权限集的应用可以彼此通信,使恶意 App 具有非法使用某些正常 App 权限的可能性。典型的攻击场景是:当一个有权限(但“困惑”)的 App 接口有漏洞时,恶意 App 利用该接口漏洞实施困惑代理攻击。

“合谋”攻击需要 2 个以上 App 之间的相互配合,又可细分为直接攻击与间接攻击两个子类,分别依赖于 App 之间的直接通信和间接通信。“合谋”攻击常常用于进行提权攻击。如攻击源 A 通过代理组件(C1、C2…)到达泄露点 P “种下种子”,如植入恶意代码。然后通过攻击源 B 对泄露点 P 进行二次攻击,实施提权操作。这样,即使检测器可以检测到攻击源 B 的攻击,攻击源 A 依旧难以被发现。

(2) 中间人攻击

这是一种在不同领域中都常见的攻击方法,在 Android 中主要攻击通过 SSL/TLS 加密的 API。原则上,App 通过 SSL/TLS 加密的 API 安全地传输敏感信息,但 App 开发者往往自行实现 SSL/TLS 证书验证程序。许多这类定制的实现故意接受所有自签名的证书,有的甚至不做任何校验就接受所有的证书,有的包含各类安全漏洞,使中间人攻击能够成功。

(3) Intent 劫持攻击

Intent 是一个包含目标组件地址和数据的消息对象,Intent 对象中主要包含 3 部分内容:动作(action)、数据和类型。对 Intent 对象中目标组件的名称或动作进行篡改,就可以实施劫持、伪造等攻击。在 Intent 中,添加数据是由 Intent 类中的 putExtra()方法完成的。考虑下述代码:

```
Intent intent = new Intent();           //创建 Intent 对象
intent.setAction("com.droider.saveinfo"); //动作
intent.putExtra("username","ABC");      //用户名
intent.putExtra("password","12345678"); //口令
startService(intent); //启动服务处理用户名和密码
sendBroadcast(intent); //发送广播处理用户名和密码
```

这段代码通过 Intent 传递需要保存的用户名和密码,然后分别通过本程序的 service 和 broadcast receiver 进行保存处理。正常情况下,该代码运行后所有的组件仅输出接收到的用户名和口令。然而,由于 Intent 中没有明确指定目的组件的名称,导致 Intent 中的数据可能被第三方

App 劫持。如果系统中同时存在多个处理该动作的组件,则 Android 系统选择启动优先级最高的组件。最先启动的 App 的组件拥有更高的优先权,这就是 Intent 劫持能够成功的原因。

(4) “类型混淆”攻击

因篇幅所限,最后,对“类型混淆(type-confusion)”攻击进行较为详细的分析。Java 最重要的一个安全问题是“类型混淆”,其背景如下。

Java 系统视对象为内存块。当 Java 程序引用某个对象时,实际上是调用一个指向存储该对象的内存地址的指针。可将指针视为具有某种类型的标签(tag),类型说明指针指向哪一类对象。Java 安全依赖于 Java 的类型安全,亦即如果 Java 是安全的,则所有的指针都有正确的标签。所谓正确的标签是指:标签与指针指向的对象的实际类型相匹配。

在类型混淆攻击中,恶意 Applet 创建两个指向相同对象但具有不相容类型标签的指针,通过一个指针写内存地址,并通过另一个指针读该内存地址。从而绕过 Java 系统的类型检查,破坏 Java 系统的安全性。

例如,下述 Java Applet 有两个指向相同内存的指针,其中一个指针标签的类型为 T,另一个指针标签的类型为 U。T 和 U 的定义如下:

```
class T {
    SecurityManager x;
}
class U {
    MyObject x;
}
```

该 Applet 运行下述代码:

```
T t = the pointer tagged T;
U u = the pointer tagged U;
t.x = System.getSecurity(); // 安全管理器
MyObject m = u.x
```

上述代码执行后,标签类型为 MyObject 的指针指向代表 Java 安全管理器对象的内存。将字段值改变为 m,即使安全管理器字段声明为“私有”,该 Applet 仍然可以修改安全管理器。虽然这个例子用于攻击安全管理器,事实上可以攻击 Java 系统的任何构件。

在 Android 的应用中,可以如下实施“类型混淆”攻击。假定有两个 Java 类。其中,PrivClass 类有一个私有的整数成员变量;ListClass 类是一种链表的实现,有一个指向下一



个成员变量的指针。函数 `set_null_to_list` 输入 `ListClass` 指针,并令该链表的下一个成员变量为空。因此,JNI 可使 `PrivClass` 实例的私有字段值为空,只需将其指针转换为 `ListClass` 指针类型的实例,并传送给函数 `set_null_to_list` 即可。

5.3 Android 恶意软件的自我保护

为了规避检测,恶意软件采用了一系列自保护的变形和隐身技术,例如代码加密、密钥置换、动态加载、代码反射、本地代码执行、代码混淆等。每种变形技术的应用也是多种多样、五花八门。例如,代码混淆技术包括:数据混淆(分割/合并变量和数据结构)、词法混淆(重命名函数和变量)、布局混淆(改变数据的分布顺序、删除调试和注释信息)、控制流混淆(改变代码的组织结构、改变代码的执行方向、改变程序的分支转移结构、伪造额外的无用控制流)、类混淆(类合并、类拆分)等。

上述变形技术可以进一步进行如下分类。

(1) 平凡的变形

无需实现代码级的变化或 `AndroidManifest` 清单文件中元数据的变化。例如:重打包、反汇编与再汇编。

(2) 通过静态分析可能发现的变形

例如:数据加密;代码重排序;无用码插入;组合变形;重命名包;重命名标识符等。

(3) 只有通过动态分析才能发现的变形

例如:字节码加密、代码反射等。

6 Android 分析与防御

一般地,Android 的安全分析可以分为静态分析和动态分析以及结合这两种方法的综合分析。

静态分析用于安装前检测,其分析流程如下:首先下载 Android 的应用程序安装包 APK,通过 `Ded`^[13]和 `Dex2Jar`^[14]等工具进行反编译,将 APK 文件反编译为 Java 源文件或

JAR 文件,然后对其进行源代码级的解析。例如,以 Dalvik 虚拟机字节码为输入,检测 App 的敏感信息是否泄露。静态分析的优点是速度快、轻量级;缺点是因为无法真实模拟程序的动态运行,存在误报问题。

表 5 列举了几种典型的静态分析方法,包括:名称、目的和分析特征^[16-23]。

在动态分析情形,需要定制和重写 Android 系统(因为开源,所以可行),包括:安全机制的改写;在原生 Android 系统中加入监视器,实时监视数据的流向;在危险函数调用时检测所需的权限,保证整个调用链上都拥有安全的权限。动态分析的优点是:检测精度较高;缺点是:需要修改 Android 系统源代码,形成用户全新定制的操作系统,同时需要通过更新 ROM 安装这种定制系统。

在动态分析中,TaintDroid^[15]是一种著名的动态脏数据检测技术:对敏感数据进行污点标记,当这些数据通过程序变量、文件和进程间通信等途径扩散时进行跟踪审查。如果污点数据在一个污点利用点(如网络接口)离开系统,TaintDroid 就在日志中记录数据标记、传输数据的应用程序和数据流向目的地,实现对多种敏感数据泄露源点的追踪。图 2 是 TaintDroid 在 Android 内的架构具体说明如下。

(1) 标记污点信息。

(2) 污点接口调用本地方法,与 Dalvik 虚拟机解释器进行交互,在虚拟污点映射图中存储指定的污点标记。

(3) Dalvik 虚拟机根据可信应用使用污点信息的数据流规则传播污点标签。每个解释器实例都同时传播污点标签。

(4) 修改后的 Binder 库保证 Parcel 有一个表示所有相关数据的组合污点标记的污点标签。

(5) Parcel 通过内核透明地传送,并被远程的不可信应用接收。

表 5 几种典型的静态分析方法

静态分析方法	目的	分析特征
SCanDroid	App 保密性与完整性保护校验	基于语言的安全模型的 App 推理
FlowDroid	敏感信息泄露检测	静态污点分析
Amandroid	组件间数据流分析	ICC 调用相关性和数据依赖性分析
Stowaway	App 最大权限集推导	API 调用识别;权限分析
PScout	生成权限映射图	反向可达性分析
ComDroid	基于 ICC 的漏洞检测	基于 Intent 的数据流分析
Chex	基于 ICC 的漏洞检测	调用图分析;数据依赖图的可达性分析
DroidChecker	敏感信息泄露检测	控制流图搜索;脏数据分析

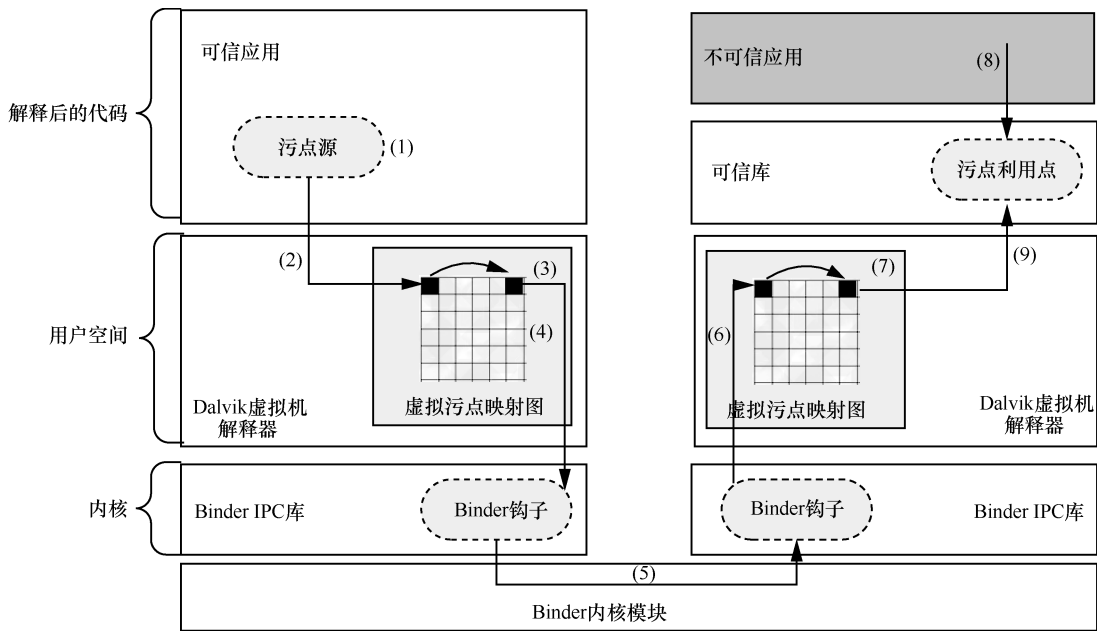


图 2 TaintDroid 在 Android 内的架构

(6)修改后的 Binder 库从 Parcel 中恢复污点标签并分配给由它读出的所有的值。

(7)远程 Dalvik 虚拟机实例在不可信应用中以相同的方式传播污点标签。

(8)不可信应用调用标记为污点利用点(如网络接口)的库。

(9)该库为有疑问的数据,恢复污点标签,并报告该事件。

在具体实现中,上述架构需要仔细处理下述问题:污点标签存储;解释后代码的污点传播;本地代码的污点传播;IPC 污点传播;辅助存储的污点传播。例如,污点标签有多种存储方法,但不同的方法有不同的性能和内存开销。

表 6 列举了几种典型的动态分析方法,包括:名称、目

的和分析特征^[24-32]。

7 现状与发展趋势

通过 8 年的应用实践,Android 的功能日益强大,其安全性也不断改善。现今,Android 的安全性研究已经进入一个新阶段,研究者从不同角度寻找 Android 现有安全保护机制的突破口,例如:针对 Android 的更新过程;针对 Android 数据残余保护机制;针对第三方库滥用宿主 App 拥有的权限;针对 Android 的传感器安全模型;针对 ICC 机制等,取得了一些新的重要成果。研究现状与可能的发展方向包括但不限于如下几个方面。

- 大多数提出的分析方法是静态的,因此只能检测出可能的漏洞与攻击。能够阻止运行时攻击的动态分

表 6 几种典型的动态分析方法

动态分析方法	目的	分析特征
AASandbox	通过系统调用追踪监控行为	系统调用
Andromaly	通过系统资源使用检测恶意软件	低层设备特征(如电池使用、CP 负载)
CopperDroid	通过系统调用追踪监控行为	系统调用
Crowdroid	通过系统调用追踪监控行为	系统调用
DroidBox	监控外部访问的沙盒	污点利用点 API 方法
DroidRanger	通过预先设定的行为脚印检测恶意软件	API 方法调用序列
DroidScope	API 追踪、指令追踪和污点追踪的插件	污点源/污点利用点 API 方法
RiskRanker	通过已知的漏洞签名检测恶意软件	API 方法调用序列
TaintDroid	检测敏感信息泄露	污点源/污点利用点 API 方法
VetDroid	通过权限使用行为检测恶意软件	可以映射到 API 方法的权限请求



析方法仍然不多。

- 提权攻击的检测方法是研究的焦点,但主要针对基于 Intent 和基于内核 IPC 的跨进程攻击,很少研究通过网络信道和通过用户输入劫持的跨进程攻击检测方法。
- 关于一个 App 内部敏感信息泄露的检测研究较为完善,但如何防止敏感信息泄露的研究还远远不够。
- 至今,所有的解决方案都存在精度不够的问题。如何进行“精准”检测是今后研究的方向。
- 所有的现有方法都没有考虑与移动设备进行交互的环境,今后应当加强研究物联网环境下的 Android 安全问题。
- 对 Dalvik 虚拟机涉及的安全问题有大量深入的研究,但鲜见关于 Android 5.0 版本后新引入的 ART 虚拟机的安全性研究。
- 对单一攻击方法的研究比较充分与深入,但对结合各种攻击方法的复合型攻击的研究还不够,应当特别加强对合谋攻击的研究。
- 对通过隐蔽通道的合谋攻击的研究才刚刚开始,今后应当加强对这种利用隐蔽通道进行合谋攻击的研究。图 3 是 3 种利用 Android 系统应用程序的隐蔽通道的例子:通过内容提供器的隐蔽通道(一方写入信息,另一方读取信息);通过广播接收器的隐蔽通道(一方触发广播,另一方读取广播);通过系统服务的隐蔽通道(一方更新设置,另一方读取设置)。

8 结束语

Android 自 2008 年 9 月 23 日正式发布以来,经历了近 8 年的发展历程,覆盖了版本 1.0~6.0(版本 7.0 即将问世)和 API 级别 1~23(级别 24 即将问世)。Android 以其开

放性和亲民的设计思路取得了巨大成功,获得全世界用户的广泛欢迎。与此同时,Android 的安全性构成了巨大的挑战。针对 Android 的攻击日益增多,攻击方法日趋先进和复杂。“道高一尺,魔高一丈”,Android 的分析技术和防御方法也日臻成熟和完善。

本文比较全面地综述了 Android 的安全性研究,内容涵盖 Android 版本更新的发展历程;Android 系统的特色(Android 系统的分层结构、Android 的组件与组件间通信、Android 的进程通信机制、程序开发语言和 JNI、Dalvik 虚拟机);Android 的安全机制;Android 的安全隐患分析(安全隐患的综合分析、具体安全隐患的分析:第三方库导致的 Android 安全隐患以及系统更新导致的 Android 安全隐患);Android 恶意软件与攻击(典型的恶意软件:“HummingBad”和“康夫机器人”、Android 恶意软件的特征与分类、Android 的典型攻击方法、Android 恶意软件的保护);Android 的分析与防御(典型的静态和动态分析方法,包括名称、目的和分析特征;典型的分析方法举例)。

因篇幅所限,本文只能对某些特定的研究成果进行较为详细的阐述,如系统更新导致的 Android 安全隐患、第三方库导致的 Android 安全隐患、“类型混淆”攻击、动态污点追踪技术等。不仅内容的选取难免挂一漏万,而且一些重要的内容也没有涉及,如加强底层操作系统的安全性措施(如通过 SELinux)、通过 ARM 架构下的 TrustZone 实现一个额外的处理器模式,从而建立一个隔离与安全的环境,保证内核安全与框架安全等。

最后,本文阐述了 Android 安全的研究现状与发展趋势以及今后可能的研究方向,供广大读者参考。

参考文献:

- [1] Gartner. Gartner report [EB/OL]. [2016-02-16]. <http://www.gartner.com/newsroom/id/3215217>.

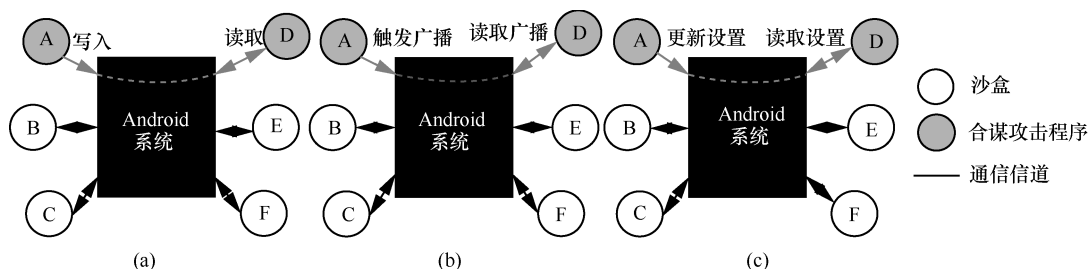


图 3 通过 Android 系统应用程序的隐蔽通道

- [2] ENCK W, ONGTANG M, MCDANIEL P. Understanding Android security [J]. IEEE Security & Privacy, 2009, 7(1): 50-57.
- [3] ENCK W, OCTEAU D, MCDANIEL P, et al. A study of Android application security[J]. British Medical Journal, 2015, 2(3859): 1175.
- [4] 卿斯汉. Android 安全研究进展 [J]. 软件学报, 2016, 27(1): 45-71.
QING S H. Research progress on Android security[J]. Journal of Software, 2016, 27(1): 45-71.
- [5] ELISH K O, SHU X, YAO D, et al. Profiling user-trigger dependence for Android malware detection [J]. Computers & Security, 2015, 49(C): 255-273.
- [6] FANG Z, HAN W, LI Y. Permission-based Android security: issues and counter measures[J]. Computers & Security, 2014(43): 205-218.
- [7] SHABTAI A, KANONOV U, ELOVICI Y, et al. Andromaly: a behavioral malware detection framework for android devices[J]. Journal of Intelligent Information Systems, 2012, 38 (1): 161-190.
- [8] ZHANG X, YING K, AAFER Y, et al. Life after app uninstallation: are the data still alive data residue attacks on Android [C]//The 23rd Network and Distributed System Security Symposium (NDSS 2016), February 21-24, 2016, San Diego, California, USA. [S.l.:s.n.], 2016.
- [9] XING L, PAN X, WANG R, et al. Upgrading your android, elevating my malware: privilege escalation through mobile os updating [C]//The 2014 IEEE Symposium on Security and Privacy (SP 2014), May 18-21, 2014, San Jose, California, USA. New Jersey: IEEE Press, 2014: 393-408.
- [10] SEO J, KIM D, CHO D, et al. FLEXDROID: enforcing in-app privilege separation in Android [C]//The 23rd Network and Distributed System Security Symp (NDSS 2016), February 21-24, 2016, San Diego, California, USA. [S.l.:s.n.], 2016.
- [11] ARP D, SPREITZENBARTH M, HUBNER M, et al. Drebin: effective and explainable detection of Android malware in your pocket [C]//The 21st Network and Distributed System Security Symp (NDSS 2014), February 23-26, 2014, San Diego, California, USA. [S.l.:s.n.], 2014.
- [12] CHECK POINT. HummingBad: a persistent mobile chain attack [EB/OL]. [2016-02-16]. <http://blog.checkpoint.com/2016/02/04/hummingbad-a-persistent-mobile-chain-attack/>.
- [13] Ded: decompiling Android applications[EB/OL]. [2016-02-16]. <http://siis.cse.psu.edu/ded/>.
- [14] Android decompiling with Dex2jar[EB/OL]. [2016-02-16]. <http://code.google.com/p/dex2jar/2015>.
- [15] ENCK W, GILBERT P, CHUN B G, et al. TaintDroid: an information flow tracking system for real- time privacy monitoring on smartphones [J]. Communications of the ACM, 2014, 57(3): 99-106.
- [16] FUCHS A P, CHAUDHURI A, FOSTER J S. SCanDroid: automated security certification of Android Applications [R/OL]. (2015-02-03) [2016-02-16]. <http://www.cs.umd.edu/~avik/papers/scandroidascaa.pdf>.
- [17] ARZT S, RASTHOFER S, FRITZ C, et al. FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps [J]. ACM Sigplan Notices, 2014, 49(6): 259-269.
- [18] WEI F, ROY S, OU X, et al. Amandroid: a precise and general inter-component data flow analysisframework for security vetting of Android apps [C]//The 21st ACM Conference on Computer and Communications Security (CCS'14), November 3-7, 2014, Scottsdale, Arizona, USA. [S.l.:s.n.], 2014: 1329-1341.
- [19] FELT A P, CHIN E, HANNA S, et al. Android permissions demystied [C]//The 18th ACM Conference on Computer and Communications Security (CCS'11), Oct 17-21, 2011, Chicago, IL, USA. [S.l.:s.n.], 2011: 627-638.
- [20] AU K W Y, ZHOU Y F, HUANG Z, et al. 2012. PScout: analyzing the Android permission specification [C]//The 19th ACM Conference on Computer and Communications Security (CCS'12), Oct 16-18, 2012, Raleigh, NC, USA. New York: ACM Press, 2012: 217-228.
- [21] CHIN E, FELT A P, GREENWOOD K, et al. Analyzing inter-application communication inAndroid [C]//The 9th International Conference on Mobile Systems, Applications, and Services (MobiSys'11), June 29-July 1, 2011, Washington, DC, USA. [S.l.:s.n.], 2011: 239-252.
- [22] LU L, LI Z, WU Z, et al. Chex: statically vetting Android apps for component hijacking vulnerabilities [C]//The 19th ACM Conference on Computer and Communications Security (CCS 2012), Oct 16-18, 2012, Raleigh, NC, USA. New York: ACM Press, 2012: 229-240.
- [23] CHAN P P F, HUI L C K, YIU S M. Droidchecker: analyzing Android applications for capability leak [C]//The 15th ACM Conf. on Security and Privacy in Wireless and Mobile Networks (WiSec 2012), April 16-18, 2012, Tucson, Arizona, USA. New York: ACM Press, 2012: 125-136.
- [24] BLASING T, BATYUK L, SCHMIDT A D, et al. An android application sandbox system for suspicious software detection[C]// The 5th International Conference on Malicious and Unwanted Software (MALWARE), Oct 19-20, 2010, Fajardo, USA. New Jersey: IEEE Press, 2010: 55-62.
- [25] SHABTAI A, KANONOV U, ELOVICI Y, et al. Andromaly: a behavioral malware detection framework for android devices[J]. Journal of Intelligent Information Systems, 2012, 38 (1): 161-190.



- [26] TAM K, KHAN S J, FATTORI A, et al. Copperdroid: automatic reconstruction of android malware behaviors [C]//The Symposium on Network and Distributed System Security (NDSS), Aug 18, 2015, San Diego, USA. [S.l.:s.n.], 2015.
- [27] BURGUERA I, ZURUTUZA U, NADJM-TEHRANI S. Crowdroid: behavior-based malware detection system for android[C]//The 1st ACM Workshop on Security and Privacy in Smart Phones and Mobile Devices, October 17, 2011, New York, NY, USA. New York: ACM Press, 2011: 15-26.
- [28] DESNOS A, LANTZ P. DroidBox: an Android application sandbox for dynamic analysis[EB/OL]. [2016-02-16]. <https://code.google.com/p/droidbox/>.
- [29] ZHOU Y, WANG Z, ZHOU W, et al. Hey, you, get off of my market: detecting malicious apps in official and alternative Android markets [C]//NDSS 2012, February 5-8, 2012, San Diego, California, USA. [S.l.:s.n.], 2012.
- [30] YAN L K, YIN H. DroidScope: seamlessly reconstructing OS and Dalvik semantic views for dynamic Android malware analysis [C]//The 21st USENIX Conference on Security Symposium, August 8-10, Bellevue, WA, USA. New York: ACM Press, 2012: 29.
- [31] GRACE M, ZHOU Y, ZHANG Q, et al. RiskRanker: scalable and accurate zero-day android malware detection [C]//The 10th International Conference on Mobile Systems, Applications and Services, June 26-29, 2012, Low Wood Bay, The Lake District, UK. New York: ACM Press, 2012: 281-294.
- [32] ZHANG Y, YANG M, XU B, et al. Vetting undesirable behaviors in Android apps with permission use analysis [C]//The 2013 ACM SIGSAC Conference on Computer & Communications Security, November 4-8, 2013, Berlin, Germany. New York: ACM Press, 2013: 611-622.

[作者简介]



卿斯汉(1939-),男,中国科学院软件研究所首席研究员,北京大学软件与微电子学院教授,主要研究方向为移动安全、可信计算、云安全、操作系统安全等。